Ethereum Classic Client (Mantis) Security Audit

Preliminary Report, 2018-01-26

FOR PUBLIC RELEASE

Contents

1 Summary

IOHK is an engineering company that builds cryptocurrencies and blockchains for academic institutions, government entities, and corporations. Major projects of IOHK are Ethereum Classic, the Daedalus wallet, and the Cardano platform.

This document reports on our security audit of IOHK's Ethereum Classic client "Mantis", a piece of software integrated in the Daedalus wallet application. The audit aimed to discover security shortcomings in Mantis and help implement relevant mitigations.

We did not find any critical security issues, but report 3 medium-severity issues and 4 low-severity issues. Mantis otherwise shows good security engineering, with secure defaults, defensive coding, and a clear code base that facilitates auditing.

The audit was lead by Dr. Jean-Philippe Aumasson, Principal Research Engineer at Kudelski Security in January 2017, and took approximately 38 hours of work.

2 Introduction

This security audit covers IOHK's Mantis Ethereum Classic client, with a focus on its cryptographic components. The audit aimed to discover security shortcomings in the protocols' logic as well as implementation errors and potential for abuse from malicious users. The main components reviewed include the keystore mechanism, network protocol implementation (RLPx), transaction validators, Ethereum virtual machine, and JSON RPC server.

The source code reviewed is in the repository `github.com/input-output-hk/mantis` in the branch phase/daedalus (commit `ef9590f`, Dec 18 2017), and includes approximately 12,000 lines of Scala code in the main source directory `src/main/scala/io/iohk/ethereum`. Most of the work consisted in static analysis of the source code. When needed, we ran simulations to verify some of our observations, leveraging Mantis' test suite.

The audit was performed by Dr. Jean-Philippe Aumasson, Principal Research Engineer, with support from Yolan Romailler, Cryptography Researcher. Approximately 38 hours of work were spent on the engagement.

We report the following findings:

- 3 medium-severity issues
- 4 low-severity issues
- 5 non-security, minor observations

All the security issues have been investigated and fixed when possible.

Disclaimer Due to time constraints, proof-of-concepts were not created for all issues found. Some findings may therefore be false alarms, or more benign than expected. But in doubt, we prefer to report too many than too few issues.

Conversely, as with any security audit, we probably missed some issues affecting the application. For example, yet unknown security issues in the Scala language, or subtle protocol edge cases that could be discovered through formal verification, for example.

Furthermore, we did not review the code of third-party dependencies, but only checked for known vulnerabilities or recently reported bugs therein. Bugs in dependencies such as `github.com/json4s/json4s` (JSON parsing), `github.com/suzaku-io/boopickle` (serialization), or BouncyCastle (crypto, ASN1 parsers, etc.) could lead to exploitable vulnerabilities, since their code directly processed untrusted data.

3 Security issues

This section enumerates the alleged security issues discovered, presenting for each issue a brief description aimed to a reader familiar with the code base, mitigation proposals, as well as a severity rating in low/medium/high.

The Status subsection will report the feedback from developers (issue correctness, mitigation implemented, etc.). When relevant, the Status subsection includes references to pull requests. We have reviewed all the pull requests cited, to validate that they correctly address the issue.

3.1 MANTIS-001: Insecure hash function used for integrity checks (MD5)

Severity: Medium

Description

`BootstrapDownload` objects check the integrity of a file downloaded from a remote URL using an MD5 digest. An attacker could forge a different, potentially malicious, file matching the expected MD5 digest.

Recommendation

Use SHA-2, SHA-3, or BLAKE2.

Status

MD5 has been replaced with SHA-512 in the pull request "[EC-416] Change bootstrap hashing to SHA512"[1].

3.2 MANTIS-002: Potentially insecure network protocol (RLPx)

Severity: Medium

Description

Mantis uses the RLPx network and transport protocol, specified at `https://github.com/ethereum/devp2p/blob/master/rlpx.md`. RLPx is the network protocol in Ethereum's DEVp2p peer-to-peer protocol, and is therefore widely used. However, to the best of our knowledge, RLPx has never been reviewed by cryptographers and isn't based on provably secure protocols, and its security properties are not clearly stated.

We didn't do a complete review of RLPx, but in limited time we noticed that the RLPx handshake is vulnerable to *key-compromise impersonation* (KCI) attacks: if a recipient's long-term key is compromised, an attacker can forge initiator messages and pretend to be the initiator (by issuing a signature of a previous static-shared-secret using a fresh ephemeral key). An attacker can even do the converse: forge a signature of a fresh static-shared-secret with the same ephemeral private key as the legitimate initiator, by picking a fresh nonce that is equal to the xor of the two shared secrets and the older nonce.

The custom MAC-based construction looks okay, but is not a standard construction let alone of provably secure one. In particular, the xoring of keys with nonces is a trick that often led to insecure schemes (and that facilitates side-channel attacks).

[1] `https://github.com/input-output-hk/mantis/pull/385`

The cryptographic primitives in RLPx have an inconsistent security level: AES-256-CTR is used for encrypting frames, whereas the handshake primitives provide 128-bit security (256-bit curve, AES-128). Finally, the key derivation scheme is inefficient, and could be performed with fewer calls to the hash function (5 currently).

Recommendation

A thorough analysis of RLPx seems necessary.

Status

Mantis developers cannot do much about it, except adhere to the protocol to ensure interoperability.

3.3 MANTIS-003: Incomplete gas values validation

Severity: Medium

Description

Here a negative value of the `gasUsed` `BigInt` in a `BlockHeader` will not be rejected as invalid:

```
1  private def validateGasUsed(blockHeader: BlockHeader): Either[BlockHeaderError,
   ↪  BlockHeaderValid] =
2    if(blockHeader.gasUsed<=blockHeader.gasLimit) Right(BlockHeaderValid)
3    else Left(HeaderGasUsedError)
```

The implications and exploitability of this are unclear.

Recommendation

A fix is simple, just check that `gasUsed` is positive.

Status

Pull request "[EC-418] Validation that gasUsed is positive number"[2] fixed the issue with the following patch to `BlockHeaderValidator.scala`:

```
1   private def validateGasUsed(blockHeader: BlockHeader): Either[BlockHeaderError,
    ↪  BlockHeaderValid] =
2   -    if(blockHeader.gasUsed<=blockHeader.gasLimit) Right(BlockHeaderValid)
3   +    if(blockHeader.gasUsed<=blockHeader.gasLimit && blockHeader.gasUsed >= 0)
    ↪  Right(BlockHeaderValid)
4        else Left(HeaderGasUsedError)
```

3.4 MANTIS-004: SHA-1 supported in ECIESCoder

Severity: Low

Description

The `decryptSimple()/encryptSimple()` methods of `ECIESCoder` use SHA-1 as a hash function, instead of SHA-256. SHA-1 collisions can be found practically, and may be exploited here.

Recommendation

Deprecate these methods, or switch to SHA-256 (truncated to 160 bits if necessary).

[2] `https://github.com/input-output-hk/mantis/pull/386`

Status

These two functions are actually only used for testing purposes.

3.5 MANTIS-005: AES implementation vulnerable to cache-timing attacks

Severity: Low

Description

The AES implementation used is that from BouncyCastle, at `https://github.com/bcgit/bc-java/blob/master/core/src/main/java/org/bouncycastle/crypto/engines/AESEngine.java`. This type of implementation is vulnerable to cache-timing attacks; an attacker running a process on the same host may be able to determine the AES secret key.

Recommendation

Using the AES from the JDK (from `javax.crypto`) seems to be a safer option. For example, OpenJDK's latest versions will use AES-NI instructions if they are available, see e.g. `https://bugs.openjdk.java.net/browse/JDK-7184394`. With such implementation, AES will also be faster than with BouncyCastle's implementation.

Status

Developers investigated the possible mitigations and noted that using AES-256 from Java versions prior to 8u161 requires the installation of Java Cryptography Extension (JCE), which would reduce the compatibility of the application. Given the threat model of Mantis, we agreed that this issue is a low risk, and that a fix can be postponed.

3.6 MANTIS-006: No password minimal size enforced

Severity: Low

Description

The method `Keystore.newAccount()` will accepted any `String` object as a valid password, even the empty string. We presume that this is designed on purpose, since the password strength check is performed by the Daedalus client ("Note that password needs to be at least 7 characters long, and have at least 1 uppercase, 1 lowercase letter and 1 number.") It would nonetheless be good practice to enforce a minimal length, as a defense layer in case of callers omitting such a check.

Recommendation

We suggest to enforce a length limit of at least eight characters.

Status

Pull request "[EC-421] Minimal passphrase length enforcement"[3] enforces a minimal length of eight characters.

3.7 MANTIS-007: Sensitive data not cleared before being garbage collected

Severity: Low

[3] `https://github.com/input-output-hk/mantis/pull/387`

Description

Sensitive data such as private keys can overwritten with zero or random data in order to prevent other processes to read it when memory is reused (this can for example be performed using the `SecureZeroMemory()` function in the Windows API). In Mantis, sensitive data includes at least private keys and passphrases, and is not cleared before being garbage collected.

Recommendation

A solution is not obvious, because most sensitive data in Mantis consists of immutable objects (`String`, `ByteString`, or `BigInteger`), which can't be zeroized. There are two solutions to this obstacle, none being ideal: first, switch to a mutable type (such as `Array` objects), but this would require a major redesign and wouldn't work for BigIntegers (althought there exists a private class `MutableBigInteger`); a second solution is to use reflection to modify immutable objects.

Status

Developers investigated possible mitigations, and noted the following points:

- Switching to mutable objects would not solve the issue, since libraries (such as Circe and Akka) used for interfaces with third-party applications use immutable objects.
- Even (e.g.) zeroized `Char[]` objects would not guarantee that all copies of the sensitive data have been zeroized, because of the way generational garbage collectors work.
- Reflection introduced complexity to the code and therefore potential bug. Too, it could slow down the application, and again would not prevent the garbage collector from creating copies of the object.
- There are other possible workarounds, none of which is really satisfactory because of the added complexity, limited effectiveness, and/or compatibility

issues:

- Using `sun.misc.Unsafe` class API, which allows direct memory access[4].
- Using `ByteBuffer.allocateDirect` to allocate memory outside the Java heap.
- Allocate larger objects to prevent the garbage collector of making copies of the objects, possibly by tweaking the value of `PretenureSizeThreshold`.

We therefore agreed that a fix can be postponed.

[4] See `https://highlyscalable.wordpress.com/2012/02/02/direct-memory-access-in-java/`

4 Observations

This section reports various observations that are not security issues to be fixed.

4.1 JSON-RPC security

A recent security issue in Parity's Ethereum client is caused by an overly permissive cross-domain domain whitelist in its JSON-RPC server (see `https://www.talosintelligence.com/reports/TALOS-2017-0508`). Mantis also runs a JSON-RPC server, but doesn't have the same vulnerability, because the whitelist is empty by default (parameter `cors-allowed-origins` in the file application.conf). This whitelist should therefore be set to only include domains authorized to access the server, and not a wildcard symbol.

4.2 ECIES decryption speed-up

The EthereumIESEngine.decryptBlock() method, as used in RLPx, decrypts data protected with an encrypt-the-MAC scheme, by first decrypting the ciphertext and then checking the MAC's value (using a constant-time byte array comparison). A disadvantage of this approach is that the ciphertext will always be decryption, even when it shouldn't—namely, when the MAC is incorrect. It would be more efficient to first check the MAC and discard the encrypted block if the MAC is incorrect.

4.3 Note on keystore encryption

The keystore mechanism uses scrypt as a password-based key derivation function, and AES-CTR as encryption primitives to encrypt a wallet's key. AES-CTR is used with a random nonce, which is often not recommended, but the secret key is random as well, which eliminates the risk of a random IV.

Legacy primitives are also supported, but not used in the current version of the code: PBKDF2-HMAC-SHA-256 and AES-CBC. If PBKDF2 is used, the number of iterations should be high enough (say, 50000) to protect the passphrases.

4.4 Different Keccak implementations

Two implementations of Keccak (SHA-3) are used in Mantis: one from scrypt (in Keystore) and one from BouncyCastle (in RLPx). It may be simpler to only use that from BouncyCastle, unless there's a specific reason not to.

4.5 Unused dependency

The following dependency declared in `build.sbt` is not used in Mantis, and can be omitted: `"org.whispersystems" % "curve25519-java" sha1 "09091eb56d696d0d0d70d00b84e6037dcd3d98b6",`

5 About

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit `https://www.kudelskisecurity.com`.

Kudelski Security
route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland